RICH MEDIA PUBLISHING

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims the benefit of priority of co-pending U.S. Provisional Patent Application Serial No. 60/466,431, entitled "Rich Media Publishing", filed April 28, 2003. Benefit of priority of the filing date of April 28, 2003 is hereby claimed, and the disclosure of the Provisional Patent Application is hereby incorporated by reference.

This application is related to co-pending U.S. Patent Application Serial No. --, entitled "Content Management for Rich Media Publishing System", filed March 30, 2004, Attorney Docket No. 450103-04598.2, and co-pending U.S. Patent Application Serial No. --, entitled "Support Applications for Rich Media Publishing", filed March 30, 2004, Attorney Docket No. 450103-04598.3. Disclosures of these U.S. Patent Applications are hereby incorporated by reference.

BACKGROUND

The rapid publication of media content is desirable for publishers intent on delivering media content faster to larger audiences. The digital representation of media content combined with computing and networking technologies now provide a powerful way to publish. According to this new mode of publishing, networking technology permits the delivery of digitized media content over a network to end user computers. Communication protocols define how the digitized media content is exchanged over the network. A media player runs on the end user computer (e.g., as software application) to allow the user to play or otherwise experience the media content.

Digital representations of media content come in different types. These types are generally defined according to a series of publishing variables which can include, but are not limited to, the file format, bit rate, communication protocol(s), physical medium, compression algorithm, and/or digital rights management information associated with the media content. The type of digitized media content used will depend upon a number of factors, such as, the

computing and/or networking technology used in the process of publishing and the nature of the content itself.

Digitized media content types can also be categorized according to the type of encoding or compression technique that is used to reduce the physical size of the media content, or according to the type of physical medium that supports the storage of the media content. Different kinds of physical medium are used in publishing media content, such as magnetic or optical storage devices, memory devices, and wireless mediums.

The emergence of a growing number of media players has created a widening gap between the richness of the various types of media content and the diverse capabilities of the client devices to handle the content. As a result, the technology selection process for the end user has become quite complicated. For example, the user often cannot be certain that a given media player will be able to play the type of media content in which he or she is interested. Also, the user may be required to frequently download new media playing software in order to access desired content.

SUMMARY

This disclosure provides system and method of publishing a media project. In one implementation, a media publishing system, includes: a network interface to connect the media publishing system to a user; a plurality of web services to enable the user to build, publish, and access a media project using templates of media items grouped into categories; and a data storage to provide a file system to said plurality of web services, where the file system allows the user to access media items.

In another implementation, a method of publishing a media project includes: selecting a category; selecting a template of media items from the category, said template of media items including a plurality of media slots, each media slot capable of receiving media items in a particular arrangement; and selecting and arranging the media items in said each media slot.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1A shows one implementation of a Rich Media Publishing environment.

Figure 2A shows an example of the layout and features of a template.

Figure 2B shows another example of the layout and features of a template.

Figure 3 shows a flowchart of one implementation of building a project.

Figure 4 shows a flowchart of one implementation of publishing a project.

Figure 5 shows a flowchart of one implementation of accessing a published project.

Figure 6 shows a main build screen for selecting a template category from a list of media categories.

Figure 7 shows a screen for selecting a project from a list of media projects for the selected category.

Figures 8A and 8B show two preview screens for a selected template.

Figure 9 shows a screen for editing the selected project selected in Figure 7.

Figure 10A is an edit screen for editing a background template.

Figure 10B is an edit screen for editing a music template.

Figure 10C is an edit screen for editing a photo template.

Figure 10D is an edit screen for editing an audio template.

Figure 10E is an edit screen for editing a video template.

Figures 11 through 17 show screen shots for selecting a source for the import file template.

Figures 18 through 23 show screen shots for selecting options in publishing a project.

Figure 24 shows a management screen for managing published projects.

Figure 25 illustrates the Model-View-Controller (MVC) architecture in accordance with one implementation.

Figure 26 illustrates the interoperability of these software components in accordance with one implementation.

Figure 27 illustrates one implementation of component model interfaces for Data Service Layer EJBs.

Figure 28A shows the base build screen for an album template category.

Figures 28B through 28E show the base build screens for journal, e-card, music player, and game template categories, respectively.

Figure 29A shows a browse templates pop-up screen.

Figure 29B shows an upload files pop-up screen.

Figure 29C shows an edit image pop-up screen.

Figure 29D shows a template background pop-up screen.

Figure 29E shows a template music pop-up screen.

Figure 29F shows a journal add entry pop-up screen.

Figure 29G shows an edit image in mask pop-up screen.

Figure 30A shows a pipeline screen for entering display name of the project.

Figures 30B and 30C show two pipeline screens for adjusting gallery settings.

Figure 30D shows a pipeline screen for entering names to share the project.

Figure 30E shows a pipeline screen for adding and/or editing the e-mail address entered in the share screen.

Figures 30F and 30G show pipeline screens for confirming and finalizing the publication of the project.

Figure 31A shows a pipeline screen for viewing a list of projects.

Figures 31B and 31C show project view screens of a published project for a premium member and a free trial member, respectively.

Figures 31D and 31E show personal gallery screens for a premium member and a free trial member, respectively.

Figure 31F shows a pipeline screen for deleting a project.

Figure 31G shows a pipeline screen for providing a HyperText Markup Language (HTML) into the project website.

Figure 31H shows a pipeline screen for changing gallery settings.

Figure 31I shows a pipeline screen for entering names to share the project.

Figure 31J shows a pipeline screen for finalizing the updated settings for the project.

Figures 32A and 32B show help screens for helping with preparation and access of the project according to one implementation.

DETAILED DESCRIPTION

This disclosure describes systems and methods that provide greater efficiency and simplicity in publication of media contents. As used herein, the term "media contents" refers to

any information, including audio, video, images, sound, data, text, other contents, or combination of these contents, which can be perceived by human senses.

Rich media publishing provides an environment for users to create and publish media projects including various types of media. Users can select various media items to be used in a project and then publish a completed project to be viewed and experienced by others.

In one implementation, a user at a desktop or laptop computer connects to a Rich Media Publishing (RMP) server system through the Internet using a web browser software application. The RMP server system presents a web site that, among other services, allows a user to build and publish an RMP project. To build a project, the user selects an RMP template. The template is a presentation framework and includes a number of media slots. Each media slot defines a genre of media (e.g., image, audio, video) and a specific target format (e.g., a JPEG format image that is 320x480 pixels).

The user selects a media item for each media slot in the selected template from a repository according to the genre of the media slot. The repository is part of the RMP server system and stores media items of various genres in various formats. The user can also upload media items to be stored in the repository and select uploaded media items for media slots. The specific format of a selected media item does not need to match the target format of the corresponding media slot because the RMP server system provides transcoding to convert a media item to the target format when presenting the project.

In addition, each template belongs to a category of templates. Templates in the same category have the same number and genres of media slots. While each template may provide a different presentation, the media slots are the same, and so the same media items can be used for templates in the same category. As a result, the user can switch among templates in the same category without changing the media item selections. Further, the RMP server system enables producers to make changes (e.g., fixes and/or updates) to the templates at one location, and allows the changes to take effect on all templates without any action taken by the user.

After the user has completed building the project, the user can publish the project. The published project is available to the user and other users to access from the RMP server system. When a user accesses the published project, the RMP server system presents the template and

the media items assigned to the media slots of the user. The RMP server system provides the media items in the target formats defined by the template and so provides appropriate transcoding. In this way, the user can share media with other users in an enjoyable and creative environment.

FIG. 1A shows one implementation of a Rich Media Publishing environment 100. An RMP server system 105 includes components providing web services 110 and data stores 115. The RMP server system 105 includes one or more network servers (not shown) linked together in a local internal network to implement these components. The web services 110 are a combination of application programs to provide the services described below. In one implementation, the web services 110 are presented to a user as a web site including a hierarchy of web pages with embedded controls. The data stores 115 include one or more data storage systems to provide database storage as well as file storage, such as in a hierarchical file system. The RMP server system 105 is connected to the public Internet 120 for communication with clients.

The web services 110 are provided by an RMP platform 112, a CORE platform (Create Once Render Everywhere) 114, and a content distribution platform 116. The RMP platform 112 allows users to manage and publish media. The RMP platform 112 includes a Member Publishing service 160, a Repository service 162, Repository Filters services 164, and Administrative services 166. The Member Publishing service 160 allows users to build RMP projects by manipulating data and media items, as well as data structures forming the RMP templates. As discussed below, templates are built by producers (developers) using development applications. Producers build templates according to guidelines set out in a Template Development Kit, distributed to producers. As discussed below, users interact with the RMP server system 105 through a client system 125 to build a project based on a selected template and selected media items. The Repository service 162 provides a repository that is a virtual file system that allows clients (users and producers) to access media items available to projects. As discussed below, from the user's point of view, the media items are presented for selection and manipulation by the user so as to appear to be stored in a hierarchical file system, while the actual organization of media items and data is hidden from the user. For example,

while the user may be shown that all the image media items are grouped together, in fact the image media items are separated by format. The Repository Filters services 164 provide a framework for performing operations on retrieved media items in the course of uploading and downloading media items. As discussed below, a filter performs one or more operations on a media item to convert the media item (non-destructively) from its original format to a format closer to or matching the target format specified by a template. Two examples of filters are an image manipulation system providing resizing of an image, and an audio transcoder for changing the format of an audio media item (e.g., from MP3 to SWF). The Administrative services 166 provide support for customer service (e.g., by allowing data and file access and manipulation by a customer service client). In one implementation, the web services 110 also support subscriptions for user and payment facilities. In addition, the web services 110 can also support different levels of subscriptions and so provide different levels of access to the services and resources of the RMP server system 105.

The CORE platform 114 provides a multi-renderer multi-language engine that allows multiple user interface (UI) representations to be derived from a single source written in IDML (Interface Definition Markup Language). As stated above, the CORE platform 114 enables Rendering services 168, UI Management services 170, Publishing services 172, and Content Management services (CMS) 174. The Rendering services 168 provide a rendering pipeline for transforming the IDML representation of a UI into a specific XML rendering language (e.g., HTML, WML) and into human readable language. The UI Management services 170 provide support for a UI Management tool used by producers for editing IDML primitives and collections. The Publishing services 172 provide a publishing pipeline that allows clients to select sources for both IDML and content data and target systems for deployment. The Content Management services 174 provide support for managing content areas in the RMP server system 105.

The CDP platform 116 provides support for identity and commerce transactions. The CDP platform 116 includes Identity services 176 and Commerce services 178. The Identity services 176 support registration and login functionality. The Commerce services 178 provide access to product listings, pricing, and promotions, and support financial transactions (e.g., credit card transactions).

A user system 125 is also connected to the Internet 120. The user system 125 communicates with the RMP server system 105 through the Internet 120. The user system 125 includes one or more end-user applications 130. The end-user applications 130 are for accessing the RMP server system 105 and uploading and downloading data and media items to build, publish, and present RMP projects. In one implementation, the end-user applications 130 include a web browser software application 126, a member publishing tool 128, a member publishing viewer 132, a web folder 134, an upload control 136, and storage tools 138. The web browser 126 is a HTTP/HTML client application. Applications from the RMP server system 105 can run in the web browser 126, such as a file access application. The member publishing tool 128 runs embedded in the web browser (e.g., as a Flash application) and allows the user to build and publish projects with the RMP server system 105. The member publishing viewer 132 runs embedded in the web browser 126 or as a separate application (e.g., as a Flash 5- or MX-based application) and allows viewing of a project. The web folder 134 provides for manipulation of media items and files stored on the RMP server system 105 using a folder-based file interface (e.g., the native Windows Explorer interface). The web folder 134 interacts with the Repository service of the RMP server system 105. The upload control 136 is a control embedded in the web browser (e.g., as an active X (win32 driven) control) and allows users to use "drag & drop" to upload files to the RMP server system 105. The upload control 136 also interacts with the Repository service. The storage tools 138 also provide support for storing media items on the RMP server system 105 (e.g., as native WIN32 applications providing Sonic Foundry tools).

A producer system 135 is connected to the RMP server system 105. In another implementation, the producer system is included within the RMP server system. In yet another implementation, the producer system is connected to the Internet and communicates with the RMP server system through the Internet. The producer system 135 includes one or more development applications 140. The development applications 140 are for accessing the RMP server system 105 to build and support the web services of the RMP server system 105, such as to build and publish templates for users to work with in building projects. In one implementation, the development applications 140 include a CORE UI management tool and a CMS tool. The CORE UI management tool supports building, editing, and publishing IDML

user interfaces. The IDML UI's can be designed to be renderer- and language-independent. The IDML UI's can then be published in multiple rendering environments. The CMS tool supports management of content written for a specific renderer. These tools support a separation of content and UI's.

A support system 145 is also connected to the RMP server system 105. In another implementation, the support system is included within the RMP server system. In yet another implementation, the support system is connected to the Internet and communicates with the RMP server system through the Internet. The support system 145 includes one or more support applications 150. The support applications 150 are for accessing the RMP server system 105 to support the web services of the RMP server system 105, such as for maintenance. In one implementation, the support applications 150 include a customer service application. A customer service representative at the support system uses the customer service application to access and control a user's files, media items, and projects.

The RMP environment allows a user to create and publish an RMP project. A user at a user system uses the end-user applications, such as a web browser and member publishing tool, to design the project. The RMP server system builds the project using the Member Publishing service. In addition, the creation of a project is supported by the uploading and selection of media items in the repository of the RMP server system. In one implementation, the RMP server system generates XML and/or HTML code for a project, including links to data and media items stored in the RMP server system. The generated code is used by client systems to present the project.

The Member Publishing service provides access to data structures implementing a collection of templates. A template has a layout and one or more visual and/or audio features (e.g., background image, background video, background music, animations, slide shows, sounds, controls, etc.). The layout and features are set by a producer that built the template (e.g., using HTML code and corresponding media items). The layout and features of a template are generally not to be changed by an end-user. A template also has one or more media slots. A media slot is an open or undefined part of the template. A media item can be assigned to each media slot. In one implementation, a media slot has a data structure (or part of

a data structure) to store a reference to an assigned media item. In one implementation, the project is a data structure that includes template data structure according to a selected template and includes a media slot data structure for each media slot in the selected template. It is these media slot data structures that indicate the assigned media items.

FIG. 2A shows an example of the layout and features of a template 200. In the template 200, a background image 205, a character body 210, and the position of the character body 210 in the template 200 are set features. The face 215 of the character is blank because the character face 215 is a media slot. A user can select an image and assign the selected image to the media slot for the face 215. FIG. 2B shows another example of a template 250. In the template 250, a rabbit character 255 has a blank face 260 representing a media slot.

A media slot has a genre and a target format. The genre indicates the type of media item that can be assigned to that media slot. For example, in one implementation, a media slot can have one of four genres: image, video, audio, or animation. Similarly, the Member Publishing service recognizes or determines a genre for any selected media item. The Member Publishing service prevents a user from assigning a media item of the wrong genre to a media slot. However, within a genre, any format of media item is acceptable. For example, for an image genre media slot, a user can select a JPG file, a GIF file, a bitmap file, or some other still image format file. The target format of a media slot indicates the format in which the template causes the media item to be requested when the media item for the media slot is to be presented. When a project is presented, the media item assigned to each media slot is retrieved and converted to meet the target format of that media slot. The conversion can be performed by the RMP server system, by the client system, or by a combination. These conversions are performed non-destructively so the original media item is preserved in the repository. In addition, these conversions are performed on the fly when the request to present a project is made. Different levels of caching of converted media items can also be provided to improve performance at the cost of storage.

The templates are grouped into categories. Templates in the same category have the same media slots but can have completely different set features. For example, referring again to FIG. 2, a second template in the same category may have a different background scene and a different character body, but still has one media slot for an image. A third template in the same

category may have completely different features including multiple image features and background music, but still has one media slot for an image. The media slots in templates in the same category also have a particular one-to-one correspondence. In another implementation, there is a hierarchy of categories. In this case, templates at a lower level have the same media slots as the template above as well as additional media slots (similar to a class and sub-class relationship).

Because templates in the same category have the same number and genres of media slots, the template can be replaced with another template in the same category without reselecting media items. Returning to the example above shown in FIG. 2, if the user selects a new template in the same category, the same image media item assigned to the image media slot in the original template will be assigned to the image media slot in the new template. In an implementation having a hierarchy of categories, a template can be switched to another template in the same specific category or to a template in a parent category at a higher level. When switched to a higher-level template, media slots that are not present in the higher-level template are dropped or rendered inactive.

From a data structure standpoint, in one implementation, the project data structure has a member for the features (the set features that do not change) of the selected template and for each media slot according to the selected template. Each media slot member indicates a corresponding media slot data structure. Each media slot data structure in turn indicates a media item. When the template is changed, the media slot data structures are not changed, but instead only data structures reflecting the features of the new template are changed. In another implementation, as discussed above, when the template is switched, the data structure for each media slot, or a reference to that data structure, is passed to the corresponding media slot in the new template (e.g., the data structure for the corresponding media slot in the new template is set to store the reference to the same media item). For example, the first media slot of the original template indicates a media item assigned to the first media slot by storing a reference to a data structure indicating the media item. When a new template is selected to replace the original template, the reference for the first media slot is passed to the new template and the first media slot of the new template is set to store that reference. As a result, the first media slot

of the new template then indicates the same media item as had been indicated in the original template. Various other data structure implementations are also possible.

A template can also have different versions for different platforms (e.g., computer web browser and phone web browser). Similar to templates in the same category, platform templates (i.e., different versions of a template for respective platforms) have the same genre and number of media slots and so are interchangeable, but the target format and characteristics of the media item requested can vary (e.g., requesting different resolution images to accommodate different display devices). Similarly, the layout and features between platform templates can be different. A project can list a single template and the initial connection and requests from the user system establish the platform of the user system and so which platform template to use. Accordingly, in one implementation, the details of the platform template are kept at the RMP server system while the code for the project on the user system is the same for each platform. In another implementation, some media slots include variable characteristics that are dependent on the characteristics of the platform. In this case, the requests for media items are customized by user system according to the characteristics of the user system.

In another implementation, a template also includes settable features. A settable feature controls an aspect of the presentation of a project such as background color or font characteristics. A settable feature does not have an assigned media item. As discussed above, media items are assigned to media slots. In one implementation, the settings for settable features are reflected in HTML code for the project built according to the template. The settable features in templates in the same category also match to facilitate seamless transition between templates. In another implementation, the settable features are unique to some or all templates and so do not carry over between templates in the same category. As described below, a user selects settings for the settable features while building the project. In another implementation, settable features are set automatically according to settings in a user profile.

FIG. 3 shows a flowchart 300 of one implementation of building a project. Before beginning, a user establishes a connection to the RMP server system and accesses the Member Publishing service (e.g., by navigating through a web site to a member publishing section). The user is presented with a collection of template categories and selects a category, block 305.

As described above, templates in the same category have the same number and genre of media slots. In one implementation, each category represents a type of presentation with each template in the category representing a particular style of that type of presentation. Examples of categories include, but are not limited to, albums, journals, scrapbooks, music players, e-cards, and games. The RMP server system can present the categories for selection in various ways, such as in a list or a representation of folders similar to or within the repository.

After selecting a category, the user is presented with a collection of templates and selects a template, block 310. The RMP server system can present the templates in various ways, such as in a list of thumbnail images, or a representation of folders similar to or within the repository. The RMP server system presents to the user one or more previews of how a project according to a template would appear. The user can preview multiple templates before selecting one template to use for the project. The user can also later change to another template. However, if the user changes to a template in a different category, some or all selections of media items may be lost.

After selecting a template, the user selects media items for the media slots in the selected template, block 315. The user selects a media item for each media slot in the template. The Member Publishing service supports selecting media items using various techniques, such as from a list, by entering a name, using drag and drop, or selecting from the file system representation of the repository. A user can select a default or recommended media item. A template can have a default media item assigned to one or more media slots and the user can select this default media item by not changing the assignment. A template can have one or more recommended media items for a media slot and the user can select one of the recommended media items, such as from a list. Default and recommended media items can also be established at a higher level than the template, such as for a category or for all templates. In another implementation, marketing information is used to recommend media items to users. Default and recommended media items can also be linked together into groups to form a set for the template. In addition, when switching between templates in the same category, when a default or recommended item has been selected in the original template, the default or recommended item for the same media slot in the new template can also be used

(although this new item may be a different media item). Default and recommended media items are stored in the repository.

A user can select a media item stored in the repository. The Member Publishing service and the Repository service present to the user a virtual file system of media items to select from. For example, the user is presented with a hierarchy of folders of different types of media items organized by the genre and content of the media items (e.g., images of celebrities are grouped together). As discussed below, the Repository service hides the actual data storage of the media items (e.g., where media items are stored as physical files on one or more servers organized by location which determined by a hash map algorithm) and instead presents the virtual file system. The repository includes media items provided by the RMP server system, media items uploaded and stored by the user (e.g., in a virtual set of folders under "My Files"), and media items uploaded by other users and made public.

A user can upload media items to the repository through the web browser application on the user's system. The web browser supports various techniques for uploading files, such as the web folder, the upload control, and the storage tools discussed above. The user can select a media item stored on the user system and cause the upload using a native file service of the user system that interacts with the web browser or by providing a pathname to the web browser. The user can also drag and drop a media item from within the GUI of the user system to the web browser. The drag and drop support allows a user to drag and drop a media item directly into the web browser window and does not require a separate window or interface as a visual target for the drag and drop operation. In one implementation, the drag and drop support is provided through an embedded ActiveX control.

A user can also select a media item stored outside the repository. The user selects a media item from a source location, such as the user system or another network system, and the media item is uploaded to the repository. A reference to the selected item is stored in the repository (e.g., as a "virtual" media item).

While the user is selecting media items, the RMP server system presents a preview of the project with the selected media items. Alternatively, the preview is presented upon demand. In this way, the user can evaluate the choices made and change selections to improve the project. The user can also change the template while selecting media items. If the new

template is in the same category as the former template, the selected media items are preserved and the user does not need to select the media items again. In an implementation supporting settable features, the user also selects settings for the settable features at this time.

As the user selects a template and media items, the user can also edit the presentation of the project, block 320. As noted above, the RMP server system presents a preview of the project. The user can adjust various aspects that the template has defined as being adjustable. The user can change the settings for settable features. The user can also adjust the presentation of media items that have an adjustable presentation style. For example, an image media slot may have an adjustable size within the template. Another type of media slot has an adjustable level of zoom or rotation for an image or an adjustable cropping portion to present different sections of the image. Another type of media slot for an audio media item has an adjustable volume or balance. The adjustments are recorded in the project and included in the request to retrieve the media item when the project is presented. The RMP server system provides the adjustments in presenting the media items as a conversion or filtering applied to the stored media item. The RMP server system returns the result to the presenting application and caches a copy of the result for future use. The original media item is not changed. Alternatively, changes can be applied by the web browser application on the user system, or a combination of changes can be applied at the RMP server system and at the user system.

As the user makes the selections and edits for the project, the RMP server system builds and updates project code for the project. The project code is code to be used by the web browser of a user to present the project to the user. The project code includes instructions according to the template and the selections and edits made by the publishing user to present the project to a user according to the publishing user's design. For some elements of the project, the project code includes instructions to the web browser on what to present, such as text or a background color. For media items assigned to media slots in the project, the project code includes requests to be sent to the RMP server system. A request indicates the target format in which the RMP server system is to provide the media item. A request can also indicate other changes or adjustments the RMP server system is to apply to the media item, such as resizing. The project code can also include instructions for the web browser for modifications to apply to the received media items, such as rotation. In one implementation,

some or all of the features to be presented as part of the project are included in the project code as references to resources or code on the RMP server system. In this way, the RMP server system provides for syndication of changes, such as when templates are updated. Any changes made to the data and template underlying a project will be provided transparently to the presented project without changing the project code. The referenced resource may change, but the reference itself can be maintained.

When the user has completed editing, the RMP server system stores the project code, block 325. The RMP server system provides storage for a user to preserve projects that have not yet been published. The user can return to the project for further editing at a later time. When the user has finished editing a project, the user can publish the project. The user can later retract a published project or a copy of a published project to the workspace area for modifications.

FIG. 4 shows a flowchart 400 of one implementation of publishing a project. After a user has finished making selections and editing the presentation of a project, the user can publish the project so that other users can access and experience the project. The user selects or enters a publishing name for the project, block 405. The publishing name is the name that will be used for other users to access the published project. In one implementation, the publishing name is part of a URL provided to users to access the published project.

The user selects a publication level for the project, block 410. The publication level indicates a range of users that will have access to the project. At a public level, any user with access to the RMP server system will be able to access the project. In one implementation, the RMP server system provides galleries of projects including a public gallery of projects that are available to users of the RMP server system. At a subscriber level, users that are subscribers to the RMP server system services will be able to access the project. At another level of access, a defined group of users can access the project. In an implementation supporting galleries, the RMP server system provides personal galleries for users where the user can define which other users are allowed to enter the user's personal gallery and access projects within that gallery. The RMP server system can also provide other levels of access, such as limited to the user

only, limited to users with an invitation from the publishing user, or limited to premium subscribers.

The user can also select a security level for the project, block 415. The security level restricts access within a publication level. For example, a user can assign a password to a project so that only users that enter the proper password can fully access the project. In one implementation, a user without the password can access a preview of the project, but cannot fully access the project. The RMP server system can support various types of security mechanisms, such as digital signatures.

The user selects how to announce the published project, block 420. The RMP server system provides an email notification system for new projects. In one implementation, the email announcement includes a URL to access the project. The recipient can then activate the URL (e.g., click on it) to proceed directly to the project. The user selects or provides one or more email addresses to receive a notification of the newly published project. In one implementation, the user provides email addresses from an email application program on the user system. In another implementation, the RMP server system maintains an email directory of subscribers to the RMP server system to facilitate notifications. In another implementation, the RMP server system provides a newsletter or news service to subscribers and the user can select whether to include a notice in the news service of the publication or not, as well as what kind of notice.

After reviewing the selections made for publication, the user confirms the selections, block 425. Once the confirmation has been received, the RMP server system stores the project code in storage for published projects and establishes access and security according to the user's selections, block 430. The RMP server system also provides notifications as indicated by the user's selections.

FIG. 5 shows a flowchart 500 of one implementation of accessing a published project. Once the project has been published, users with the appropriate level of access and providing the appropriate security can access the project. A user accesses the RMP server system and selects a project to experience, block 505. A user can browse through public collections or galleries of projects or personal galleries that are open to the user. A user can also request a

particular project, such as by name. If a user has received a URL for a project, the user can access the project directly using the URL without navigating through the RMP server system web site.

After selecting the project, the web browser at the user's user system downloads a copy of some or all of the project code for the project, block 510. The project code is the code built and stored by the RMP server system when the publishing user built the project. The project code includes instructions for the web browser to present the project including downloading media items. The web browser does not necessarily immediately download all the media items assigned to the project. The web browser can request and download the media items on demand as the project is presented.

As media items are to be presented in the project according to the user's actions and the project code, the web browser requests media items from the RMP server system, block 515. The project code includes a media item request for each media item assigned to the project. A media item request indicates the media item, the target format, and any modifications the RMP server system is to apply to the media item according to the characteristics of the settings of the media slots. As discussed above, media items can be presented as they are in storage, in a different format, or adjustments can be applied to the media item. For example, an image media slot in the project that has a target format of JPEG sends a request to the RMP server system for the media item to be presented as a JPEG media item. If the media item assigned to that media slot is in fact a different format, such as a GIF media item, the RMP server system applies a filter to convert the GIF item to a JPEG item and sends the JPEG item to the requesting user system. The request can also include additional adjustments to the media item, such as resizing.

When the RMP server system receives a media request from the web browser, the RMP server system retrieves the indicated media item and applies the appropriate modifications, block 520. The RMP server system maintains a cache of generated adjusted media items. The RMP server system first checks if there is already a copy of a media item matching all aspects of the request in the cache. If so, the RMP server system does not need to perform further modifications and returns a copy of the cached item. If not, the RMP server system retrieves the original media item from the repository. The RMP server system checks if the media item

is in the format requested as the target format. If not, the RMP server system applies a filter to generate a new copy of the media item in the target format. If the request includes additional modifications to be applied to the media item, such as resizing, the RMP server system applies the additional modifications to the filtered media item to match the request. The RMP server system returns the adjusted media item to the user system and stores a copy in the cache.

In one implementation, the RMP server system also checks the cache for partial matches to the request. In this case, if the cache includes an item that matches part of the request without additional modifications, the RMP server system retrieves that item and applies the remaining modifications to match the request. For example, where the request is for a JPEG image at a particular resolution, the RMP server system checks the cache and determines that there is not a JPEG image at that resolution for the indicated media item in the cache, but determines that there is a JPEG image at a different resolution for the indicated media item in the cache. Instead of retrieving the original media item, which may be in a different format and could require a filter to be in JPEG format, the RMP server system applies an adjustment to change the resolution of the cached item to match the request, creating another copy in the cache. The RMP server system returns the new adjusted media item matching the request. The RMP server system can use a priority system to determine which partial matches to use. In another implementation, the RMP server system determines whether it is better (e.g., faster) to undo one or more changes that have been applied to a cached copy and then work forward rather than starting from the original media item or from a cached copy that fewer changes applied.

In one implementation, the RMP server system provides the media item to the web browser through a secure connection or using security measure to control access. The RMP server system can use encoding for some or all media items sent to the user system (e.g., encoding items that are not free). The RMP server system can also use access links that have a limited lifespan so that a request for the same media item will need to use a new access link after a period of time. As part of accessing the project, the web browser receives and updates the access links to maintain fresh links.

After the user system has received the media item matching the request from the RMP server system, the web browser applies any additional changes indicated by the project code to

the received media item, block 525. As discussed above, the project code can indicate that some modifications to a media item are to be applied by the web browser rather than at the RMP server system. For example, the project code may indicate that, after the web browser receives the media item from the RMP server system, the web browser is to apply changes such as rotation, clipping, brightness adjustment, or volume adjustment. The web browser follows a similar pattern to download and adjust any other media items as indicated by the project code.

After applying any local adjustments, the web browser presents the project and media items to match the project code, block 530. Some aspects of the project do not require media items to be downloaded and are generated at the user system, such as fonts and colors. As the user interacts with the project through the web browser, the project code may indicate modifications are to be applied resulting in local adjustments made by the web browser, new media item requests for the RMP server system, or combinations thereof.

In another implementation, layout information and features of the project other than media items are also stored as requests in the project code so that these items are also downloaded from the RMP server system. In this way, changes can be made at the RMP server system to templates or media supporting templates (e.g., background images or music) and these changes will appear in all projects using the templates as a result of the requests being sent back to the RMP server system.

In one implementation, the web browser and target formats used for media items protect the media items from unwanted copying (unwanted from the publishing user's or content owner's point of view). In one implementation, the rendering functionality of the web browser does not present any copy or save facility to a user. So the user can view the images and video of a presentation, but the web browser does not directly support preserving the media items. A similar approach can be used with audio media items. In this way, it is difficult for users to make unauthorized copies of media items presented in projects. This extra security may make it more desirable for content providers and users to upload or provide access to content to be included in projects.

In one implementation, the RMP server system provides a code publishing service. A user can download a copy of the project code for a project and include that project code in any

network accessible location. The user (or another user) can then access the stored project code and experience the presentation without downloading the project code from the RMP server system. The RMP server system will still be involved in the presentation because the media items assigned to the project are stored on the RMP server system, as well as some features of the project (e.g., template layout).

In one implementation, the RMP server system supports searching. A user can search for media items, templates, or other data available through the RMP server system. The searching can apply to data provided by the RMP server system or to data uploaded to the RMP server system by the user, another user, or a content provider.

In one implementation, the searching returns results that are context sensitive. In one implementation, the search engine filters the results according to a user profile. Some media items available to the users are not available for free. A user profile may specify a price limit for search results. Some media items are available to users according to their subscription level. A user profile may indicate only to return search results that are at a certain subscription level. Alternatively, the search engine may return only those results that are at the user's subscription level.

In another implementation, the search engine provides a preview feature. The search engine allows a user to select a search result and retrieve a preview of the indicated item, such as a thumbnail of a template or image, or a short segment of a video or audio item. This preview may be provided for a pay item before the user purchases the media item for inclusion in a project.

As discussed above, the repository is a storage system for the media items available to and used in projects built and published through the RMP server system. The repository can be implemented as one or more storage devices using one or more databases. In one implementation, each type of media item is stored in a respective storage device (e.g., JPEG images on one storage device, MP3 files on a second storage device, streaming media on another storage device, and so on). The repository maintains information about the stored media items to present a virtual hierarchical file system to the user so that the user can easily

access, move, add, and delete (etc.) media items according to a hierarchical organization. The user can create and delete folders in the repository and the RMP server system will present the media items according to these folders even though the actual storage organization in the repository may be quite different. In addition, each user can have a different view of the data in the repository as the users build different file structures, such as through the creation and modification of virtual folders. Users can also customize the presentation features, such as types of folders, colors, etc.

The RMP server system also stores data in one or more databases and/or files to support the operation of the web site. This data is referred to herein as producer data. Producer data includes data to implement the templates, other than the media item data stored in the repository. Producers can access and modify media data in building and modifying templates.

The RMP server system is supported by a client-side Content Management Services (CMS) tool. The CMS tool is for content management to allow producers to access and manage the producer data and any media data needed for development. The CMS tool works with a multi-environment publishing pipeline to control access to and distribution of producer data as it is being developed. For example, one publishing pipeline supports three environments: a developing/production environment, a staging environment, and a live environment. Each environment maintains a separate store of data. When a producer begins work on a template (for example), the producer creates and edits the template and associated data in the development environment. When the producer has completed the development work, the producer migrates the template and data to the staging environment. A complete copy of the data is brought to the data storage of the next environment. In the staging environment, the template and data are tested and reviewed. When the template and data are approved, they are migrated to the live environment. Again, a complete copy of the data is brought to the data storage of the next environment. In the live environment, users can access the template and data for member publishing. Back-end constraints are provided to insure that database conflicts are avoided from environment to environment.

In one implementation, producers are restricted to accessing data only in the development environment. Any changes made are submitted as updates up through the

pipeline. As updates are moved upward, the updates are applied to the versions of data stored for each environment. In one implementation, the entire data set is not migrated upward, but only the data or files that have been marked as modified.

In another implementation, a different number of environments are supported, such as two, or four, or more. The number is customized to the goals of the RMP server system, producers, and users.

In another implementation, the RMP server system provides simultaneous support for publishing of database items and physical files, using the same user interface protocols. To a producer using CMS, database items are published in the same manner, e.g., with the same "PUBLISH/DELETE" flags for all items shown in the UI. The list of physical files to be published is constructed using a method that traverses both source and destination file system trees, producing a list of differences that is displayed to the producer as files to be PUBLISHED or DELETED. This list is a selectable list of publishable physical files. In one implementation, the publishable physical files may be distributed in several different locations, such as in the repository of the RMP server system and/or in the local storage of the user system.

In another implementation, a producer can preview data through the CMS. A producer can preview a specific piece of producer content or a specific physical file directly from the content management system (e.g., style sheets and JSPs, allowing a WYSIWYG preview of *ML content). In one implementation, the CMS tool provides a "Preview" button or supports a "double-click" of a particular item (e.g., for previewing items listed in the "Content" or "Tips and Tricks" tabs in the illustrated figures below). The preview automatically launches the appropriate application or browser needed to see the content/file, and automatically performs the necessary transformations on internal markup language strings, using configuration settings and paths, allowing that content to reference other content elements correctly. The producer can then avoid having to access the content separately using the end-user rendering system.

Referring again to FIG. 1, it was disclosed that the user system 125 includes one or more end-user applications 130, which are used for accessing the RMP server system 105. The end-user applications 130 include a web browser software application 126, a member

publishing tool 128, a member publishing viewer 132, a web folder 134, an upload control 136, and storage tools 138. The member publishing tool 128 runs embedded in the web browser (e.g., as a Flash MX-based application) and allows the user to build and publish projects with the RMP server system 105. The RMP server system 105 includes components providing web services 110 and data stores 115.

In one implementation, the user interface for the member publishing includes a Flash application, which enables the users to create unique flash-based presentations including their personal media. The application enables users to upload their media into a number of predefined templates and then send or allow access to these completed projects to friends and family. The user interface is developed in Flash to enable execution of business logic on the client side as well as develop a more robust user interface. In general, much of the logic is on the client side, and the interface with the server side is enabled via the web tier, which includes Flash Remoting Gateway, Controller Enterprise JavaBeans (EJB), and application processors. Thus, the web tier makes the application's business logic available on the Internet.

FIG. 6 through FIG. 24 illustrate screen shots of various user interfaces for the member publishing tool in accordance with one implementation. For example, FIG. 6 shows a main build screen 600 for selecting a template category from a list of media categories. FIG. 7 shows a screen 700 for selecting a project from a list of media projects for the selected category. In the illustrated implementation, the selected category is an elegant album. FIG. 8A and FIG. 8B show preview screens 800, 810 for a selected template. FIG. 9 shows a screen 900 for editing the selected project selected in FIG. 7.

FIG. 10A through FIG. 10E show edit screens 1000, 1010, 1020, 1030, 1040, respectively, for editing templates. FIG. 10A is an edit screen 1000 for editing a background template. FIG. 10B is an edit screen 1000 for editing a music template. FIG. 10C is an edit screen 1000 for editing a photo template. FIG. 10D is an edit screen 1000 for editing an audio template. FIG. 10E is an edit screen 1000 for editing a video template. FIG. 11 through FIG. 17 show screen shots 1100-1700 for selecting a source for the import file template. FIG. 18 through FIG. 23 show screen shots 1800-2300 for selecting options in publishing a project. FIG. 24 shows a management screen 2400 for managing published projects. In one

implementation, the files imported from many different sources (e.g., from the user hard drive, from the RMP server repository, and/or from clips & effects). Furthermore, the files of different types can be imported transparent to the user.

As mentioned above, in one implementation, the user interface for the member publishing tools is enabled via the web tier. The web tier development for member publishing system should be designed to achieve efficient processing of requests, and secure handling of personalized information. Moreover, the web tier design should be extensible, scalable, and fault tolerant, and should be capable of supporting multiple types of thick clients, including but not limited to Flash MX and Windows applications.

To accomplish the above-described web tier development goals, the member publishing system uses a hybrid Model-View-Controller (MVC) architecture. The following paragraphs describe what the MVC architecture attempts to achieve, the available variations of the architecture, and how and why this implementation uses a hybrid approach to Model 2 to meet the aforementioned goals.

The Model-View-Controller architecture is an architectural approach for interactive applications. The architecture divides functionality among objects involved in maintaining and presenting data to minimize the degree of coupling between the objects. The architecture maps traditional application tasks--input, processing, and output--to the graphical user interaction model. The architecture also maps into the domain of multi-tier, web-based enterprise applications.

The MVC architecture divides applications into three layers--model, view, and controller--and decouples their respective responsibilities. Each layer handles specific tasks and has specific responsibilities to the other areas.

A model represents business data and business logic or operations that govern access and modification of this business data. Often the model serves as a software approximation to real-world functionality. The model notifies views when it changes and provides the ability for the view to query the model about its state. The model also provides the ability for the controller to access application functionality encapsulated by the model.

A view renders the contents of a model. The view accesses data from the model and specifies how that data should be presented. The view updates data presentation when the model changes. The view also forwards user input to a controller.

A controller defines application behavior. The controller dispatches user requests and selects views for presentation. The controller interprets user inputs and maps them into actions to be performed by the model. In a stand-alone GUI client, user inputs include button clicks and menu selections. In a web application, inputs include HTTP, GET, and POST requests to the web tier. The controller selects the next view to display based on the user interactions and the outcome of the model operations. An application typically has one controller for each set of related functionality. Some applications use a separate controller for each client type, because view interaction and selection often vary between client types.

FIG. 25 illustrates the Model-View-Controller (MVC) architecture in accordance with one implementation. The illustrated implementation depicts the relationships between the model, view, and controller layers of an MVC application.

Separating responsibilities among model, view, and controller objects reduces code duplication and makes applications easier to maintain. The separation also makes handling data easier, whether adding new data sources or changing data presentation, because business logic is kept separate from data. Further, it is easier to support new client types, because it is not necessary to change the business logic with the addition of each new type of client.

Overall structure is the most important consideration in a web-tier design. Both the sample application and the various existing web application frameworks implement some form of "Model 2" architecture, where a servlet manages client communication and business logic execution, and presentation resides mainly in Java Server Pages (JSP).

Model 1 and Model 2 refer to the absence or presence (respectively) of a controller servlet that dispatches requests from the client tier and selects views. A Model 1 architecture includes a web browser directly accessing web-tier JSP. The JSP access web-tier JavaBeans that represent the application model, and the next view to display (e.g., JSP, servlet, HTML page, and so on) is determined either by hyperlinks selected in the source document or by request parameters. A Model 1 application control is decentralized, because the current page being displayed determines the next page to display. In addition, each JSP or servlet processes

its own inputs (parameters from GET or POST). In some Model 1 architectures, choosing the next page to display occurs in scriptlet code, but this usage is considered poor form. Software maintenance on Model 1 is extremely expensive when constant updates are in place, since the programmer has to sift through piles of views to alter the site flow.

A Model 2 architecture introduces a controller servlet between the browser and the JSP pages or servlet content being delivered. The controller centralizes the logic for dispatching requests to the next view based on the request URL, input parameters, and application state. The controller also handles view selection, which decouples JSP pages and servlets from one another. Model 2 applications are easier to maintain and extend, because views do not refer to each other directly. The Model 2 controller servlet provides a single point of control for security and logging, and often encapsulates incoming data into a form usable by the back-end MVC model. For these reasons, the Model 2 architecture is recommended for most interactive applications.

An MVC application framework can greatly simplify implementing a Model 2 application. Application frameworks such as Apache Struts include a configurable front controller servlet, and provide abstract classes that can be extended to handle request dispatches. Some frameworks include macro languages or other tools that simplify application construction.

The Model 1 architecture can provide a more lightweight design for small, static applications. The Model 1 architecture is suitable for applications that have very simple page flow, have little need for centralized security control or logging, and change little over time. Model 1 applications can often be refactored to Model 2 when application requirements change.

The Model 1 architecture is best when the page navigation is simple and fixed, and when a simple directory structure can represent the structure of the pages in the application. Such applications usually embed the page flow information in the links between the pages. The presence of a forward in a JSP page implies that logic embedded in the page is making a decision about the next page to display.

In a situation where there is a need for centralized security control and logging, the information being returned is a value object encapsulating model data rather than the

presentation of a view, and the client is capable of performing its own logic, Model 2 provides a better basis from which to work.

The role of the Model 2 controller servlet will be partially fulfilled by the Flash MX remoting gateway, with the remaining functionality subsumed by a stateful EJB. The return from this EJB will be a value object rather than a view. The presentation of the view based upon the value object will be handled by the thick client. Additionally, some of the manipulation of model data may be done on the client before being communicated back to the Controller for processing. Further, the decoupling of the Controller into a servlet component and an EJB component make it possible for future client communication protocols to be easily handled.

As discussed above, in one implementation, the web tier includes a Flash MX remoting gateway, controller EJBs, and application processors. These software components work together to accomplish personalized data management per user and thick-client interaction for application production.

FIG. 26 illustrates the interoperability of these software components in accordance with one implementation. In the illustrated implementation of FIG. 26, the initial HTTP request is made of the Flash MX remoting gateway 2602. The request is passed along to the stateful EJB 2604 for user tracking and delegation. The delegated request 2606, 2608, 2610, or 2612 is processed by an application processor component 2620. A consolidated value object is then returned to the user 2600 with the operation's results.

A servlet pipeline is not used in this architecture because the bulk of the work to be done is not handled by servlets. The majority of pre- and post-processing requests can be handled by the thick-client. Although, in this case, a pipelining approach is a good general-case architectural choice, the approach may act as a detriment to performance when identifiable uses are minimal.

The Flash MX remoting gateway is a 3rd-party servlet that provides a means for Flash MX movies to communicate with server-side objects with minimal programming. Previous versions of Flash could make HTTP requests for data retrieval, including XML. The data returned would then need to be converted into a format that could be understood by the Flash

ActionScript by the ActionScript programmer. This was especially true with XML, as the support provided was limited and rather problematic.

With Flash MX, the capabilities have been expanded to now fully support XML data at a native data type. Also, it understands a proprietary format that allows serializeable Java objects to be automatically treated as ActionScript objects for direct use by the Flash programmer. The remoting gateway is the piece that handles communication between the client movie and the server side to accept incoming HTTP requests and to pass back results that are immediately available as native objects.

In one implementation, the gateway allows access to servlets, Java beans, serializeable Java objects, and EJBs. To access these entities, the client must specify the gateway URL, a connection endpoint for the service to be accessed, the method to be called on the service, and the parameters that will be supplied to the method call. The servlet endpoint is given as a context root for the servlet, the serializeable Java objects' endpoint is given as a fully-qualified classpath, and the EJB endpoint is its JNDI entry. It is for security reasons that it is preferable to use EJBs as the underlying controller for this application, because no additional information about the code structure or deployment environment is made available by examining an in-the-clear HTTP request.

In instances when sending method calls and parameters in the clear is not desirable, the requests may be made transparently over HTTPS. It is planned that login/logout requests will be handled over this protocol. While all communications could be done in this manner, it is projected that the performance hit for all users would outweigh the minute incremental security risk that someone could reverse-engineer the Flash MX protocol and attempt to inject malicious requests into the stream.

The Application delegator is a stateful EJB that acts as the other half of the Controller. It provides the request flow with a centralized location for common data handling, and manages the logic in charge of delegating actual request handling.

FIG. 27 illustrates one implementation 2700 of component model interfaces for Data Service Layer EJBs. This implementation indicates that an application processor, regardless of whether the request is made by an end-user, creator, or in-house producer, can represent each

action that may be requested by a client. One processor may handle one or more actions. Each action affects state by committing such changes to a backing repository, though the action implementation itself is not stateful.

These processors may be implemented as Java beans, plain Java objects, or stateless EJBs. The main responsibility for each such processor is to handle appropriate call parameters and perform their task as efficiently as possible. Because the backing repository manipulation is a key component of the performance, object creation and exception handling needs to be kept thin.

Referring again to FIG. 3, once the user enters the Member Publishing service, the user is presented with a collection of template categories. As described above, examples of categories include, but are not limited to, albums, journals, scrapbooks, music players, e-cards, and games. FIG. 28A through FIG. 28E show screen shots of base build screens in accordance with one implementation of building a project. Each screen shot of the illustrated implementation shows an example of a template. For example, FIG. 28A shows the base build screen 2800 for an album category template. FIG. 28B through FIG. 28E show the base build screens for journal 2830, e-card 2860, music player 2870, and game 2880 category templates, respectively.

Referring again to FIG. 28A, as an example, the base build screen 2800 for the album category template includes a change template button 2802, a change background button 2804, a change music button 2806, a media library screen 2808, a help button 2810, an album title area 2812, a media item stage 2814, an edit image button 2816, a music player object 2818, a launch player button 2820, a trash area 2822, a preview button 2824, and a publish button 2826.

The change template button 2802 enables the user to change the template. The change background button 2804 enables the user to change the background of the template. The change music button 2806 enables the user to change the template music. The media library screen 2808 displays media-related items, such as recent downloads, clips & effects, and a list of media files. The media files include image, video, music, and other related multimedia files. Moreover, the user can drag and drop items from the media library screen 2808 onto the media item stage 2814. The user can reorder and add titles and captions to the items on the media item stage 2814. The user can enter a title for the project on the album title area 2812. The edit

image button 2816 can be used to edit an image. The music player object 2818 enables playing of the music files. The launch player button 2820 launches an external multimedia player. The preview button 2824 enables the user to preview the project. The publish button 2826 prepares the project for publishing by collecting information related to publishing.

FIG. 28B shows the similar base build screen 2830 for the journal category template, which includes similar items including a change template button 2832, a change background button 2834, a change music button 2836, a media library screen 2838, a help button 2840, a journal title area 2842, a media item stage 2844, an edit journal entry button 2846, a trash area 2848, a preview button 2850, and a publish button 2852. The edit journal entry button 2846 is used to navigate to journal add/edit entry screen.

FIG. 29A through FIG. 29G show screen shots of pop-up screens in accordance with one implementation of building a project. FIG. 29A shows a browse templates pop-up screen 2900; FIG. 29B shows an upload files pop-up screen 2910; FIG. 29C shows an edit image pop-up screen 2920; FIG. 29D shows a template background pop-up screen 2930; FIG. 29E shows a template music pop-up screen 2940; FIG. 29F shows a journal add entry pop-up screen 2950; FIG. 29G shows an edit image in mask pop-up screen 2960. The pop-up screens may appear in response to actions taken in the base build screens 2800, 2830, 2860, 2870, 2880, or in response to actions taken in other pop-up screens 2900, 2910, 2920, 2930, 2940, 2950, 2960.

The browse templates pop-up screen 2900 allows the user to browse through the template category and to select a template. The upload files pop-up screen 2910 allows the user to drag and drop files on the upload files area for uploading media files. The edit image pop-up screen 2920 allows the user to edit the selected image. The template background pop-up screen 2930 allows the user to select a template background. The template music pop-up screen 2940 allows the user to select a template background music. The journal add entry pop-up screen 2950 allows the user to enter journal entry. The edit image in mask pop-up screen 2960 allows the user to edit the image in a mask.

Referring again to FIG. 4, after the user has finished making selections and editing the presentation of a project, the user can publish the project so that the user and/or other users can experience the project. FIG. 30A through FIG. 30G show screen shots of a user publish pipeline in accordance with one implementation of publishing a project. The user publish

pipeline includes a pipeline for publishing the project built according to the above-described process in connection with FIGS. 28 and FIGS. 29.

FIG. 30A shows a pipeline screen 3000 for entering display name of the project. The display name of the project also includes the Uniform Resource Locators (URL) of the project file. FIG. 30B and FIG. 30C show two pipeline screens 3010, 3020 for adjusting gallery settings, which include selections for the gallery area (*i.e.*, the personal or the public galleries), whether to show or hide the gallery, and the password. FIG. 30D shows a pipeline screen 3030 for entering names to share the project. In one example, the project can be shared by sending out an e-mail invite to the e-mail address listed in the share screen. FIG. 30E shows a pipeline screen 3040 for adding and/or editing the e-mail address entered in the share screen. FIG. 30F and FIG. 30G show pipeline screens 3050, 3060 for confirming and finalizing the publication of the project.

Referring again to FIG. 5, once the project has been published, users with the appropriate level of access and capable of providing the appropriate security can access the project. FIG. 31A through FIG. 31J show screen shots of an access pipeline in accordance with one implementation of accessing a project. The access pipeline includes a pipeline for accessing the project published according to the above-described process in connection with FIGS. 30.

FIG. 31A shows a pipeline screen 3100 for viewing a list of projects. According to the illustrated implementation, the published projects can be viewed, shared, edited, and/or deleted, while the unpublished projects can be edited or deleted. The individual project can be viewed directly by entering the URL of the project. The pipeline screen 3100 also includes a button 3102 for viewing a personal gallery. FIG. 31B and FIG. 31C show project view screens 3110, 3120 for a published project. FIG. 31B is a project view screen for a premium member. FIG. 31C is a project view screen for a free trial member, which includes a clickable area to allow the free trial member to join the premium pay membership. FIG. 31D and FIG. 31E show personal gallery screens 3130, 3140 for a premium member and a free trial member, respectively. FIG. 31F shows a pipeline screen 3150 for deleting a project. FIG. 31G shows a pipeline screen 3160 for providing a HyperText Markup Language (HTML) into the project website. FIG. 31H shows a pipeline screen 3170 for changing gallery settings. FIG. 31I shows

a pipeline screen 3180 for entering names to share the project. In one example, the project can be shared by sending out an e-mail invite to the e-mail address listed in the share screen. FIG. 31J shows a pipeline screen 3190 for finalizing the updated settings for the project.

FIG. 32A and FIG. 32B shows help screens for helping with preparation and access of the project according to one implementation.

The various implementations of the invention are realized in electronic hardware, computer software, or combinations of these technologies. Most implementations include one or more computer programs executed by a programmable computer. For example, referring to FIG. 1, in one implementation, the server system includes one or more computers executing software implementing the RMP environment. In general, each computer includes one or more processors, one or more data-storage components (e.g., volatile or non-volatile memory modules and persistent optical and magnetic storage devices, such as hard and floppy disk drives, CD-ROM drives, and magnetic tape drives), one or more input devices (e.g., mice and keyboards), and one or more output devices (e.g., display consoles and printers).

The computer programs include executable code that is usually stored in a persistent storage medium and then copied into memory at run-time. The processor executes the code by retrieving program instructions from memory in a prescribed order. When executing the program code, the computer receives data from the input and/or storage devices, performs operations on the data, and then delivers the resulting data to the output and/or storage devices.

Various illustrative implementations of the present invention have been described. However, one of ordinary skill in the art will see that additional implementations are also possible and within the scope of the present invention. For example, while the above description describes computers connecting to servers through the Internet, additional variations and combinations are possible. For example, in other implementations, different types of network-enabled devices can be used, such as a network-enabled television or phone. Accordingly, the present invention is not limited to only those implementations described above.